http://blog.csdn.net/qq\_19079937/article/details/51909259

lua中的#是获取table中有序序列的个数。例如：

**[plain]** [view plain](http://blog.csdn.net/qq_19079937/article/details/51909259) [copy](http://blog.csdn.net/qq_19079937/article/details/51909259)



1. local t = {a = 1, 1,3,4,5,6}
2. print(#t)--5

在table的实现中，有两种方式保存元素，一种 数组，一种hash散列表。#就是获取table中的数组元素个数。例如上面代码就是获取数组的个数。其中a = 1保存在散列表中。如果在table中增加几个nil，会有什么问题呢？

**[plain]** [view plain](http://blog.csdn.net/qq_19079937/article/details/51909259) [copy](http://blog.csdn.net/qq_19079937/article/details/51909259)



1. local t = {a = 1, 1,3,nil,4,5,6,nil}
2. print(#t)--2

是不是相当诡异？为什么会这样？我们从源码一步步开剥开迷雾。

**[cpp]** [view plain](http://blog.csdn.net/qq_19079937/article/details/51909259) [copy](http://blog.csdn.net/qq_19079937/article/details/51909259)



1. **const** TValue \*luaH\_getint (Table \*t, lua\_Integer key) {
2. /\* (1 <= key && key <= t->sizearray) \*/
3. **if** (l\_castS2U(key) - 1 < t->sizearray)
4. **return** &t->array[key - 1];
5. **else** {
6. Node \*n = hashint(t, key);
7. **for** (;;) {  /\* check whether 'key' is somewhere in the chain \*/
8. **if** (ttisinteger(gkey(n)) && ivalue(gkey(n)) == key)
9. **return** gval(n);  /\* that's it \*/
10. **else** {
11. **int** nx = gnext(n);
12. **if** (nx == 0) **break**;
13. n += nx;
14. }
15. }
16. **return** luaO\_nilobject;
17. }
18. }

**[cpp]** [view plain](http://blog.csdn.net/qq_19079937/article/details/51909259) [copy](http://blog.csdn.net/qq_19079937/article/details/51909259)



1. **static** **int** unbound\_search (Table \*t, unsigned **int** j) {
2. unsigned **int** i = j;  /\* i is zero or a present index \*/
3. j++;
4. /\* find 'i' and 'j' such that i is present and j is not \*/
5. **while** (!ttisnil(luaH\_getint(t, j))) {//获取数组最后一个元素是否为nil，
6. i = j;
7. **if** (j > cast(unsigned **int**, MAX\_INT)/2) {  /\*防止溢出，只有j > MAX\_INT/2条件才为真。说明table是一个大的table \*/
8. /\* table was built with bad purposes: resort to linear search \*/
9. i = 1;
10. **while** (!ttisnil(luaH\_getint(t, i))) i++;//当数组中出现值为nil，就终止循环。
11. **return** i - 1;
12. }
13. j \*= 2;
14. }
15. /\* 使用二分法搜索。进过上面的while循环后，i = j, j = 原来的2倍。m = (i + j)/2 我们可以理解成m = (j + 2j)/2,第一次二分法查找，m = j + floor(j/2),此时m > j(大于table数组个数)，ttisnil一定为nil，j = m.第二次二分法查找，此时i= j的，所以 m = (j + floor(j/2) + j)/2 等效于 m = j + floor(j/4)，任然大于table数组的个数,ttisnil一定为nil，j = m。经过几次这样的操作，m就等于数组的长度了\*/
16. **while** (j - i > 1) {
17. unsigned **int** m = (i+j)/2;
18. **if** (ttisnil(luaH\_getint(t, m))) j = m;
19. **else** i = m;
20. }
21. **return** i;
22. }

j++保证j是hash部分的第一个值，从j开始,如果j位置是有值的，那么将j扩大两倍，再检查两倍之后hash表中是否可以取到值，直到找到没有值的地方，这个值就在i 到 j这个区间中。然后再用折半查找找到 i 到 j之间找到的最后一个nil的，前面的就是它的长度了。 错略看来。luaH\_getint用来取值   
const TValue \*luaH\_getint (Table \*t, int key)而它的声明看来 ，第二个参数是key，通过key来取value， 而外面对传入的key是++的操作 可知计算长度用来寻找的这个key一定是个整形，而且还得是连续的(不一定)。（当然这个是不深究细节实现错略看下来的分析。。。。。）

int luaH\_getn (Table \*t) {

unsigned int j = t->sizearray;

if (j > 0 && ttisnil(&t->array[j - 1])) { **// 数组最后一个元素为空，则只计算数组部分，否则还会算hash部分**

/\* there is a boundary in the array part: (binary) search for it \*/

unsigned int i = 0;

while (j - i > 1) { **// 这里只会计算数组部分了，不会计算hash部分**

unsigned int m = (i+j)/2;

if (ttisnil(&t->array[m - 1])) j = m;

else i = m;

}

return i;

}

/\* else must find a boundary in hash part \*/

else if (t->node == dummynode) /\* hash part is empty? \*/

return j; /\* that is easy... \*/

else return unbound\_search(t, j);

}

二分法介绍：[点击我](http://blog.sina.com.cn/s/blog_a74f39a20101elgr.html)

诡异的原因知道了，就要尽量避免在table的有序数组最后插入nil值，如果二分法中间位m,恰巧m-1也是nil就会有诡异的结果。

**1）数组最后一个元素是否为空的情况**

local tab = {1, 2, 3, nil, nil, 6, nil, **8**, [9] = 9, [10] = 10, [11] = 11, [12] = 12}

print(#tab)



local tab = {1, 2, 3, nil, nil, 6, nil, **nil**, [9] = 9, [10] = 10, [11] = 11, [12] = 12}

print(#tab)



local tab = {1, 2, 3, 4, 5, 6, 7, nil, [9] = 9, [10] = 10, [11] = 11, [12] = 12}

print(#tab)

local tab = {1, 2, 3, nil, nil, 6, 7, nil, [9] = 9, [10] = 10, [11] = 11, [12] = 12}

print(#tab)

local tab = {1, 2, 3, [4] = 4, [5] = 5, 6, 7, nil, [9] = 9, [10] = 10, [11] = 11, [12] = 12}

print(#tab)



local tab = { 1, 2, 3, [5] = 10, [6] = 100 }

print(#tab)

local tab = { 1, 2, 3, 4, [5] = 10, [6] = 100 }

print(#tab)

